Likewise, FIG. **3** is a graph #__**300** logically representing a two-processor multi-processor system #__**300**. The graph #__**300** of FIG. **3** is fully connected. When communication faults occur dividing the system #__**300** into the graph #__**500** of FIG. **5**, each processor #__**112** marks the other as unreachable in the mask of reachable processors and applies the split-brain avoidance methodology described above. The processor **1**, for example, may notice its failure to receive an IamAlive message from processor **2**. The processor **1** accordingly initiates a regroup operation. In Stage I of that Regroup operation, the processor **1** starts its internal timer, resets its connectivity matrix C and suspends I/O activity. The processor **1** then sends a Regroup message and prepares to receive and compare Regroup messages in order to update its connectivity matrix C. In this scenario, however, the processor **1** receives no such Regroup messages. When the appropriate time limit has been reached (and if the processor **1** of itself constitutes enough resources to continue operations, if appropriate), the processor **1** proceeds to Stage II.

In Stage II, the processor **1** selects itself as the tie-breaker processor #__**112** since it was the lowest numbered processor #__**112** at the end of the last regroup operation to complete.

The processor **1** then applies the split-brain avoidance methodology: The processor **1** recognizes that the group of processors #__**112** of which it is a part has neither more nor less than one-half of the processors #__**112** that were present before the regroup operation began. Its group has exactly one-half of the pre-existing processors #__**112**, and the processor **1** uses the fact that it is itself the tie-breaker processor #__**112** as the decision point to continue operations.

Not being the tie breaker, the processor **2** attempts to check the state of the tie-breaker processor **1** (in one embodiment, using the service processors). If the state of the tie breaker can be determined, the processor **2** realizes that the tie breaker is healthy. The processor **2** halts.

Where the state of the tie-breaker processor **1** cannot be determined, the processor **2** checks the mask of unreachable processors. Noting that the tie breaker is marked unreachable, the processor **2** assumes that the tie breaker is healthy and halts.

Thus, the tie-breaker processor **1** continues operation while the processor **2** halts.

The processor **1** selects itself as the tie-breaker processor #__**112** and remains in Stage II until a reasonable amount of time passes. (The processor **2** cannot and indeed does not send Regroup messages as the communication fault has occurred and the processor has halted.)

The processor **1** applies the pruning process and determines the group of processors #__**112** that are to survive the regroup operation. Using its memory-resident connectivity matrix C as input, the tie breaker computes the set of all dead processors, {**2**}, and converts its matrix C into canonical form. This conversion leaves a 1×1 matrix C including only the processor **1**. The tie breaker computes the set of disconnects as the set {(**1**, **2**), (**2**, **1**)}, with D=2. However, as the set of live processors {**1**} does not include the processor **2**, applying these disconnects to that set has no effect. The number of maximal, fully connected graphs is one, and the tie breaker sets its pruning result variable to indicate that only it will survive. The tie breaker communicates this result in its subsequent Regroup messages and thus passes through Stages III and IV. The system #__**500** completes the regroup operation and continues operations with only the processor **1** running.

Finally, consider again the logical multi-processor systems #__**200**. Now, the processor **2** experiences a corruption of its time list, fails to receive timer expiration interrupts and loses its ability to send the requisite IamAlive messages. The detection of the missing IamAlive messages by any of the other processors **1** or **3–5** causes a regroup operation to begin.

In Stage I of the regroup operation as related above, the processors **1–5**, operating according to one embodiment of the invention, each refrain from sending respective Stage I Regroup messages until each receives a timer expiration interrupt. Thus, the processors **1** and **3–5** readily proceed to send Stage I Regroup messages.

By hypothesis, the processor **2** does not receive timer interrupts and thus never sends a Stage I Regroup message. The other processors **1** and **3–5** update their respective KNOWN_STAGE_1 variables #__**750***a* (and/or their respective connectivity matrices C) to reflect the healthiness of the processors **1** and **3–5** and the apparent death of the processor **2**. After some predetermined amount of time has passed waiting for the processor **2**, the processors **1** and **3–5** proceed to Stage II.

In Stage II, the processors **1** and **3–5** now broadcast Stage II Regroup messages. The processors **1** and **3–5** are healthy and the processor **2** is still malatose, and the Stage II Regroup messages eventually reflect this condition. The KNOWN_STAGE_2 variable #__**750***b* becomes equal to the KNOWN_STAGE_1 variable #__**750***a*.

The processor **2**, by hypothesis, still receives the Regroup messages from the processors **1** and **3–5**. It eventually receives a Stage II Regroup message wherein the KNOWN_STAGE_1 and _2 variables #__**750***a*, #__**750***b* are equal and exclude the processor **2**. The processor **2** notices this type of Stage II Regroup message and halts.

Processors **1** and **3–5** proceed through the remainder of the regroup operation and form the system N__**200**'. Now, instead of the IamAlives missing from the processor **2** periodically perturbing the system N__**200**, the system N__**200**' excludes the processor **2** altogether. (Also, the processor **2** is dead and therefore harmless.)

Of course, the program text for such software incorporating the invention herein disclosed can exist in its static form on a magnetic, optical or other disk; in ROM, in RAM or in another integrated circuit; on magnetic tape; or in another data storage medium. That data storage medium may be integral to or insertable into a computer system.

What is claimed is:

1. In a multi-processor having a plurality of processors, each of said plurality of processors having a respective memory, a method for improving tolerance for inter-processor communications faults, said method comprising:

communicatively coupling said plurality of processors;

then detecting a communications failure;

then attempting to firstly determine on each of said plurality of processors still operating which of said plurality of processors are still operating and still communicatively coupled, thereby determining said each processor's respective view of said multi-processor system;

then secondly determining on said each processor still operating whether said each processor still operating is to halt operations if said each processor's respective view of said multi-processor system indicates that said processor(s) still operating and still communicatively coupled number less than one-half of said plurality of processors; and

then continuing or halting operations on said each processor according to said second determination.